

1330.1020

APPENDIX

Object: n_cst_busabs_activity
Process:180

Method: of_AssessCharges()

Main Routine. This function is the starting point for the charge calculation process. It is called after the terms of the activity have been successfully verified. After gathering information required to determine and calculate charges on the activity, this function instantiates the AssessCharges service object which performs the actual calculation and assignment of charges.

```
////////////////////////////////////
/*
  Function:      of_AssessCharges
  Access:       public
  Arguments:    none
  Returns:      SUCCESS if successful, FAILURE if not
  Description:  Checks if the activity status is one that allows charges to
be               performed and if so, kicks off the process.
                  Call of_AssessCharges on the Assess Charges service object.
                  This will kick off the logic to copy held charges and
generate         new charges for this activity.
instrument,      The service object uses the handles to the current
                  activity, and list of charges to add items to to gather the
                  charges for the given scenario.
*/
////////////////////////////////////
//
// Revision History
//
// edelaski      2 jan 1997      Initial version
// bschuetzler 12 aug 1997      implement charges recalc
```

```

//
//
// Copyright © 1997 American Management Systems. All rights reserved.
//
//
n_cst_bussrv_assesschrgs Inv_AssessChargesService
n_cst_busabs_instrumentterms Inv_terms
string ls_uoid, ls_Status, ls_termsuoid
boolean lb_ErrorsFound

lb_ErrorsFound = FALSE
of_enableactivitymgr(TRUE)
ls_uoid = of_GetObjectID()

// Get the current status.
ls_Status = inv_AttributeMgr.of_GetAttribute('current_status')

// If the status is verified, assess the charges.
If ls_Status = gk.Verified Then
    // Check if this is a terms type activity. If so, get terms characteristics
    if inv_activitymgr.of_newterms() then
        ls_termsuoid = inv_attributemgr.of_getattribute('active_terms_results')
        Inv_terms = inv_immediateparent.dynamic of_getcomponenthandle('terms_list'+ls_termsuoid+')
        Inv_terms.of_getinstmchrgchar(as_attributename[], as_attributevalue[])
    End If
    // Get activity type specific attributes
    inv_activitymgr.event ue_getchrgcharacteristics(as_attributename[], as_attributevalue[])
    // Assess the charges.
    Inv_AssessChargesService = CREATE n_cst_bussrv_assesschrgs
    Inv_AssessChargesService.of_setrequestor(this)
    If Inv_AssessChargesService.of_AssessCharges(as_attributename[], &
        as_attributevalue[]) < > gk.Success Then
        inv_err.of_error('UNABLE_TO_ASSESS_CHARGES', ls_uoid)
        lb_ErrorsFound = TRUE
    End If
    DESTROY Inv_AssessChargesService
    // Set recalc to NO
    If inv_AttributeMgr.of_SetAttribute('charges_recalculation_indicator', gk.No) < > gk.Success Then
        inv_err.of_error('UNABLE_TO_SET_CHARGES_RECALC_IND', ls_uoid)
        lb_ErrorsFound = TRUE
    End If
Else
    inv_err.of_error('ACTIVITY_NOT_VERIFIED', ls_uoid)
    lb_ErrorsFound = TRUE
End If

If lb_ErrorsFound Then
    Return gk.Failure
Else
    Return gk.Success
End If

```

Object: n_cst_busabs_activity

Process:182

Method: of GetChargeInfo()

This function pulls together information (e.g., attributes) from the Activity object which will be used during the charge calculation and assessment process.

////////////////////////////////////
 //////////////////////////////////////

/*

Description:

Retrieves activity-specific characteristics needed for charge preloads and calculations.

Arguments:

as_attributename[] - reference array of attributes retrieved
 as_attributevalue[] - reference array of attribute values

Returns:

None.
 string gk.FAILURE or gk.SUCCESS

*/
 //////////////////////////////////////

// Revision History

Name	Date	Description
henryln	2 Sep 1997	Initial release

////////////////////////////////////

// Copyright © 1997 American Management Systems. All Rights Reserved.

//
 //////////////////////////////////////
 //

string ls_value[]
 integer li , li_row, li_UpperBound

li_row = UpperBound(as_attributename[])
 li_row++

// Gets Close Instrument Indicator
 as_attributename[li_row] = 'close_instrument_indicator'
 as_attributevalue[li_row] = inv_activity_of_GetComponentAttribute('payment',
 'close_instrument_indicator')
 li_row++

as_attributename[li_row] = 'payment_made_indicator'

inv_Instrument_of_GetComponentAttribute('activity_list', {'activity_type',
 'current_status'}, ls_value[])

// Assigns Payment Made Indicator based on Payment Activity Status

```
li_UpperBound = UpperBound(ls_value[])
FOR li = 1 TO li_UpperBound step 2
  If ls_value[li] = gk.PAYMENT THEN
    IF ls_value[li+1] = gk.RELEASED THEN
      as_attributevalue[li_row] = gk.Yes
    ELSE
      as_attributevalue[li_row] = gk.No
    END IF
  EXIT
END IF
NEXT
```

Return gk.SUCCESS

Object: n_cst_busobj_dlcterms

Process: 184

Method: of_GetChargeInfo()

Called by the Activity business object to gather the attributes on the Terms object (and its components) that are required to determine and calculate charges for the Activity.

```

////////////////////////////////////
////////////////////////////////////
/*
Description:
    Retrieves all the information necessary for charges to determine which
    charge types to preload.

Arguments:
    as_attributename[] reference - populates all the characteristic names
    as_attributevalue[] reference - populates all the characteristic values

Returns:
    Return value
*/
////////////////////////////////////
//Revision History
//
// Name      Date      Description
// A. Name    24 oct 96    initial release
// N. Felix   28 Jan 98    Removed setting of revolve indicator b/c it is done
on
//                                superlc terms object
//
////////////////////////////////////
//
// Copyright © 1996 American Management Systems. All Rights Reserved.
//
////////////////////////////////////
/* Determine the values of various attributes and association of an LC_type
instrument
/* that are used for charge preloads.
int li_attribcount, i
string ls_partyvoid
long ll_partyrow

n_cst_busobj_collateralterms inv_collateral

li_attribcount = Upperbound(as_attributename[]) + 1

// Get Charges for
as_attributename[li_attribcount] = 'charges_for_party'
as_attributevalue[li_attribcount] =
inv_attributemgr.of_getattribute('charges_for')
li_attribcount++

// Determine collateral

as_attributename[li_attribcount] = 'collateralized_indicator'
if
gnv_app.inv_string.of_isempty(inv_attributemgr.of_getattribute('collateral_ter
ms')) then
    as_attributevalue[li_attribcount] = gk.NO
    li_attribcount++
Else
    as_attributevalue[li_attribcount] = gk.YES
    li_attribcount++
/* if collateral, determine collateral type
inv_collateral = of_getcomponenthandle('collateral_terms')

```

```

    as_attributename[li_attribcount] = 'collateral_type'
    as_attributevalue[li_attribcount] = inv_collateral.of_determcollattype()
    li_attribcount++
End If

//Operative indicator
as_attributename[li_attribcount] = 'operative_indicator'
as_attributevalue[li_attribcount] =
inv_attributemgr.of_getattribute('operative_indicator')
li_attribcount++

//Determine RMB indicators
inv_partylist = of_getcomponenthandle('party_list')
as_attributename[li_attribcount] = 'reimbursing_bank_indicator'
if inv_partylist.of_locate('party_type','RMB',ls_partyuoid, li_partyrow) then
    as_attributevalue[li_attribcount] = gk.YES
    li_attribcount++
    as_attributename[li_attribcount] = 'reimbursing_bank_charges_for_party'
    as_attributevalue[li_attribcount] =
of_getcomponentattribute('lcreimburseterms','charges_for')
    li_attribcount++
Else
    as_attributevalue[li_attribcount] = gk.NO
    li_attribcount++
End If

// Get purpose type
as_attributename[li_attribcount] = 'purpose_type'
as_attributevalue[li_attribcount] =
inv_attributemgr.of_getattribute('purpose_type')
li_attribcount++

// Get available by type
as_attributename[li_attribcount] = 'available_by_type'
as_attributevalue[li_attribcount] =
inv_attributemgr.of_getattribute('available_by')
li_attribcount++

// Get out confirm type
as_attributename[li_attribcount] = 'our_confirm_type'
as_attributevalue[li_attribcount] =
inv_attributemgr.of_getattribute('our_confirm_type')
li_attribcount++

// Get Cover Full Indicator
as_attributename[li_attribcount] = 'advise_type'
as_attributevalue[li_attribcount] =
inv_attributemgr.of_getattribute('cover_full_details_indicator')
li_attribcount++

// Get Previously Confirm Indicator
as_attributename[li_attribcount] = 'previously_confirmed_indicator'
as_attributevalue[li_attribcount] = of_getprevconfindicator()
li_attribcount++

// Get the 7 user defined attributes
For i = 1 to 7
    as_attributename[li_attribcount] =
'lc_type_instruments_user_defined_term_' + string(i)
    as_attributevalue[li_attribcount] =
inv_attributemgr.of_getattribute(as_attributename[li_attribcount])
    li_attribcount++
Next

Return gk.SUCCESS

```

Object: n_cst_bussrv_assesscharges

Process: 186

Method: of_AssessCharges()

Called by the Activity business object to assess the charges for the Activity. It first deletes all of the previously calculated charges (if any) and then calls the of_GenerateNewCharges function which calculates the charges for the activity and stores them on it as component objects (e.g., Activity_Charge)

```

////////////////////////////////////
////////////////////////////////////
/*
    Function:          of_assesscharges
    Access:            public
    Arguments:         anv_Activity          The calling activity
                      as_characteristicname[] instrument characteristic
names
                      as_characteristicvalue[] instrument characteristic
values
    Returns:           string
                      (gk.SUCCESS or error code)
    Description:       Attaches activity charges to the activity that calls this
                      responsibility.
                      To do this, it goes through the following steps:
                      - delete all old charges in the list
                      - copy charges held from a previous activity
                      - generate charges resulting from this activity
*/
////////////////////////////////////
//
// Revision History
//
// Version
// 1.0   Initial version
//
////////////////////////////////////
// Copyright © 1996 American Management Systems, Inc. All Rights Reserved.
//
////////////////////////////////////

string ls_return, ls_product, ls_producttype, ls_activitytype

// Initialize instance variables so that every function has a handle to
current instrument,
// and activity.

inv_Instrument = anv_Activity.of_GetImmediateParent()
inv_Activity = anv_Activity
ii_NumCharges = 0

// remember who called me to share error handling, user id, and password:
this.of_SetRequestor(inv_Activity)

ls_product = inv_instrument.of_getattribute('product')
ls_producttype = inv_instrument.of_getattribute('product_type')
ls_activitytype = inv_activity.of_getattribute('activity_type')

// determine whether to actually recalculate charges by first checking if the
last save
// detected a need to recalc, and if not, ask the activity if its components
and related

```

```

// objects have been changed such that a recalculation is needed. If not,
quit.
if inv_activity.of_GetAttribute('charges_recalculation_indicator') <> gk.YES
then
    if not inv_activity.of_NeedToRecalc(gk.CHARGES, {ls_product,
ls_producttype, ls_activitytype}) then
        return gk.success
    end if
end if

// insert the required characteristics into the array needed for the rule
search:
of_addcharacteristic('activity_type', ls_activitytype,
as_characteristicname[], as_characteristicvalue[])
of_addcharacteristic('product_type', ls_producttype, as_characteristicname[],
as_characteristicvalue[])
of_addcharacteristic('product', ls_product, as_characteristicname[],
as_characteristicvalue[])

// get the active terms' uoid for future reference (should be activity's
responsibility):
is_TermsUoid = inv_Activity.of_GetAttribute('active_terms_results')
if gnv_app.inv_string.of_isempty(is_TermsUoid) then
    is_TermsUoid = inv_Instrument.of_GetAttribute('active_instrument_terms')
end if

// Determine the 'Charges For' party type for future reference:
of_RetrieveChargesForPtyType()

// Request current Bank Org from activity (needed for rates / conversion /
instruction search):
is_OpBankOrg = inv_Activity.of_DetermineOperationalBkOrg()

// Delete old charges...
ls_return = of_deletecharges()
IF ls_return <> gk.SUCCESS THEN return ls_return

// copy held charges from previous activities:
ls_return = of_copyheldcharges()
IF ls_return <> gk.SUCCESS THEN return ls_return

// generate new ones based on this activity:
ls_return = of_generatenewcharges(as_characteristicname[],
as_characteristicvalue[])
IF ls_return <> gk.SUCCESS THEN return ls_return

return gk.SUCCESS

```

Object: n_cst_bussrv_assesscharges

Process: 188

Method: of GenerateNewCharges

This function determines which charges to apply to the activity by identifying the Charge Preload Rules that apply and then determining the applicable Charge Instruction (based upon the Preload Rule). For each Charge Instruction, it calls the of_CreateCharge function to add an Activity Charge component object to the Activity.

```

////////////////////////////////////
/*
  Function:      of_GenerateNewCharges
  Access:       protected
  Arguments:    as_CharacteristicName[]
                as_CharacteristicValue[]

                List of attributes/associations/derived attributes
passed         in by the activity based on which to search for
matching       charge preload rules.

  Returns:      string
                (gk.SUCCESS or gk.FAILURE)

  Description:
*/
////////////////////////////////////
// Revision History
//
// Version
// 1.0 Initial version
//
////////////////////////////////////
// Copyright © 1996 American Management Systems, Inc. All Rights Reserved.
//
////////////////////////////////////

n_cst_busobj_chrgprldrulelc inv_ChrgPrldRule
n_cst_busobj_chrginstruction inv_ChrgInstruction

string ls_RuleUoids[], ls_InstrucUoid, ls_RetrieveParms[3], ls_av[]
integer li, li_RuleCount, li_PtyCount, li_NumInstructions = 0
boolean lb_errorsfound = false

// get all party types and associated customers the instrument is currently
// aware of:
inv_Instrument.of_RetrievePartyInfo(inv_activity.of_getobjectid(),
{'party_type', 'party_customer'}, ls_av[])
li_ptycount = upperbound(ls_av[]) / 2
for li = 0 to li_ptycount - 1
  is_InstrumPtyType[upperbound(is_InstrumPtyType[])+1] = ls_av[2*li+1]
  is_InstrumPtyCustomer[upperbound(is_InstrumPtyCustomer[])+1] =
ls_av[2*li+2]
end for

// retrieve charge preload rule candidates for this product
type/product/activity type combo:
inv_ChrgPrldRule = CREATE n_cst_busobj_chrgprldrulelc
li_RuleCount = inv_ChrgPrldRule.of_cls_FindMatches(as_characteristicname[],
as_characteristicvalue[], ls_RuleUoids[])
DESTROY inv_ChrgPrldRule

// Call FindInstruction and create a charge if an instruction is found.
FOR li = 1 to li_RuleCount
  inv_ChrgPrldRule = CREATE n_cst_busobj_chrgprldrulelc

```



```

    IF Inv_ChrgPrldRule.of_open(ls_RuleUoids[li], is_userid, is_password,
gk.ACCESS_READ) = gk.SUCCESS THEN
        ls_InstrucUoid = this.of_FindInstruction(Inv_ChrgPrldRule)
        IF NOT gnv_app.inv_string.of_IsEmpty(ls_InstrucUoid) THEN
            Inv_ChrgInstruction = CREATE n_cst_busobj_chrginstruction
            IF Inv_ChrgInstruction.of_open(ls_InstrucUoid, is_userid,
is_password, gk.ACCESS_READ) = gk.SUCCESS THEN
                li_NumInstructions++
                of_createcharge(Inv_ChrgPrldRule, Inv_ChrgInstruction)
            END IF
            DESTROY Inv_ChrgInstruction
        END IF
    END IF
    DESTROY Inv_ChrgPrldRule
END FOR

// the only charge that does not get preloaded by a rule is nostro interest:
of_createnostrointerest()

return gk.success

```

Object: n_cst_busobj_chrgprldrulc

Process: 192

Method: of_cls_FindMatches

This function determines which Charge Preload Rules apply to the Activity (based upon characterizing of the Activity and its related terms) and passes back their Ids.

```

/////////////////////////////////////////////////////////////////
/*
  Function:          of_cls_FindMatches
  Access:           public

  Arguments:
  as_CharacteristicName[]  An array of instrument characteristic names
  as_CharacteristicValue[] An array of instrument characteristic names
  as_ruleUOID[]           An array of UIDs of the found rules
                          (populated by this function)

  Returns:          integer
                    The amount of rules that fire for the given
                    characteristics.

  Description:      This function receives an array of instrument
characteristics    (possible "IF" criteria of a charge preload rule) and finds
                    the firing charge preload rules whose UOIDS it passes back
in                  the second parameter.

                    The function is called by the activity charge list to find
                    instances of preload rules that the list object will then
use                to find corresponding charge instructions.

                    The function may also be called from an external object
                    (UI), e.g. for charge preload rule maintenance.
*/
/////////////////////////////////////////////////////////////////
//
// Revision History
// Version
// 1.0 Initial version
//
/////////////////////////////////////////////////////////////////
// Copyright © 1996 American Management Systems, Inc. All Rights Reserved.
/////////////////////////////////////////////////////////////////

int li_total, li_found = 0
long ll_rulerows[]
n_cst_objcls_readonlylist lnv_SearchList
string ls_Args[3]

// get the required characteristics product/product type/activity type for the
// of_open of the search list which will limit the instances to be searched:
ls_Args[1] = of_FindCharacValue('product', as_characteristicname[],
as_characteristicvalue[])
ls_Args[2] = of_FindCharacValue('product_type', as_characteristicname[],
as_characteristicvalue[])
ls_Args[3] = of_FindCharacValue('activity_type', as_characteristicname[],
as_characteristicvalue[])

// open the search list passing it the required characteristics in Args[]:
lnv_SearchList = CREATE n_cst_objcls_readonlylist
lnv_SearchList.of_Prepere('n_cst_busobj_chrgprldrulc', 'search')
IF lnv_SearchList.of_Open(ls_Args[], is_UserID, is_Password, gk.ACCESS_READ,
li_total) = gk.SUCCESS THEN
  li_found = lnv_SearchList.of_Locate(as_characteristicname[],
as_characteristicvalue[], TRUE, as_ruleuoid[], ll_rulerows[])
END IF
DESTROY lnv_SearchList
return li_found

```

Object: n_cst_bussrv_assesscharges

Process: 194

Method: of_FindInstruction()

Determines the Charge Instruction to be used for the specified Charge Preload Rule. Additional information, such as the Customer and the Operational Bank Organization is used in the determination process.

```

////////////////////////////////////
/*
  Function:      of_FindInstruction
  Access:       public
  Arguments      anv_ChrgPrldRule      Charge Preload Rule for which
matching                               instruction is to be found.

  Returns       String                  Found instruction UOID or '' (not
found)

  Description:   Search for a charge instruction for this charge preload
rule's          charge party or parties and charge type given the current
instrument      and activity by calling the 'determine_instruction' class
responsibility  on chrg_instruction.
*/
////////////////////////////////////
//
// Revision History
//
// bschuetzler   7 jan 1997      Initial version
//
////////////////////////////////////
//
// Copyright © 1997 American Management Systems. All rights reserved.
//
////////////////////////////////////

string ls_RulePtyType[], ls_InstrumPtyType[], ls_InstrumPtyCustomer[],
ls_InstrucUoid[], &
ls_Customer, ls_Instrument, ls_ChargeType, ls_searchallind,
ls_values[]
integer li, li_NumInstrucs, li_Row, li_FoundPos, li_ptycount
n_cst_busobj_chrginstruction lnv_chrginstruction

// Obtain a customer uoid for the instruction search from the first party of
those specified
// on the preload rule that exists in the instrument:

// get all party types specified on the charge preload rule:
anv_ChrgPrldRule.of_GetAttribute({'charge_party_1', 'charge_party_2',
'charge_party_3'}, ls_RulePtyType[])
ls_searchallind =
anv_ChrgPrldRule.of_GetAttribute('party_instruction_indicator')

// get all party types and associated customers the instrument is currently
aware of:
inv_Instrument.of_RetrievePartyInfo(inv_activity.of_getobjectid(),
{'party_type', 'party_customer'}, ls_values[])
li_ptycount = upperbound(ls_values[]) / 2
for li = 0 to li_ptycount - 1
  ls_InstrumPtyType[upperbound(ls_InstrumPtyType[])+1] = ls_values[2*li+1]
  ls_InstrumPtyCustomer[upperbound(ls_InstrumPtyCustomer[])+1] =
ls_values[2*li+2]
end for

// Now search for the first occurrence of a rule party in the terms parties:
li = 1

```

```

SetNull(ls_Customer)
DO WHILE li <= UpperBound(ls_RulePtyType[]) AND isNull(ls_Customer)
  FOR li_Row = 1 TO UpperBound(ls_InstrumPtyType[])
    IF ls_RulePtyType[li] = ls_InstrumPtyType[li_Row] THEN
      ls_Customer = ls_InstrumPtyCustomer[li_Row]
      EXIT
    END IF
  END FOR
  li++
LOOP

IF gnv_app.inv_string.of_IsEmpty(ls_Customer) THEN ls_Customer = ''

// Request instrument UOID from the current instrument (needed for instruction
search):
ls_Instrument = inv_Instrument.of_GetObjectID()

// Get the charge type from this rule (additional attribute in instruction
search):
ls_ChargeType = anv_ChrgPrldRule.of_GetAttribute('charge_type')

return of_findinstruction(is_opbankorg, ls_customer, ls_instrument,
ls_chargetype)

// OVERLOADED Version of of_FindInstruction is below:

n_cst_busobj_chrginstruction inv_chrginstruction
string ls_instrucuid[]

// Call the class responsibility to find an instruction based on the obtained
information:
inv_chrginstruction = create n_cst_busobj_chrginstruction
// Pass a handle to the instrument so the subsequent search doesn't open it
again:
inv_chrginstruction.of_setrequestinginstrument(inv_instrument)
inv_chrginstruction.of_cls_determineinstruction(ls_instrucuid[], TRUE, &
as_oporguoid, '', '', as_customeruoid, as_instrumentuoid,
as_chargetype)
destroy inv_chrginstruction

IF upperbound(ls_instrucuid[]) > 0 THEN
  return ls_instrucuid[1]
ELSE
  return ''
END IF

```

Object: n_cst_busobj_chrginstruction

Process: 198

Method: of_cls_DetermineInstruction()

Called by of_FindInstruction on
n_cst_bussrv_assesscharges to search for a charge
instruction for the specified operational bank org,
customer, and charge type.

/*

Function:	of_cls_determineinstruction	
Access:	public	
Arguments:	as_InstrucUoid[] ab_StopWhenFound	Found instruction Uoids (output) TRUE if search should end after an
instruction is		found, FALSE if the search should
continue to the		next level and search for more
general instructions.	as_OpBkOrgID	Oper.Bank Org. to base search on
(input)	as_CountryType	Country to base search on (input)

(input)	as_InstructionGroup	Instruction Group to base search on
	as_CustomerID	Customer to base search on (input)
	as_InstrumentID	Instrument to base search on (input)
argument value	as_AddlAttrValue	Optional additional retrieval

Returns: instruction uoids (if found), instruction parent class (if found)

Description: Using Instrument (optional), Customer (mandatory), and OpBankOrg (mandatory) derive the customer's Instruction Group and Instruction Country and the OperBankOrg's parent bank org. Then "of_Open", the instruction search list passing it today's date, the resulting six object Uoids, and an additional optional search criterion as retrieval arguments for the search list object.

one the six The call to of_Open will find all instructions that are for deactivated and possible parents and that are currently effective (not not future dated given today's date).

attempt to Now apply the common instruction search method which is to specialized find the applicable instruction by searching from the most to the most general instructions in the following order. Unless ab_StopWhenFound is FALSE, end the search when the first instruction is found. Otherwise keep going up the chain and add instruction Uoids to the output array if more applicable instructions are found.

1. Instrument
2. Customer
3. Instruction Group
4. Country
5. Operational Bank Organization
6. Parent Bank Organization

```

*/
//
// Revision History
//
// BSchuetzler 2 jan 1997 Initial version
//
//
// Copyright © 1997 American Management Systems. All rights reserved.
//
//

```

```

string ls_ParentBkOrg, ls_RetrievalArgs[], ls_TermsUoid, &
      ls_InstrumentUoid, ls_CustomerUoid, ls_OpBkOrgUoid
integer li_NumFound, li, li_EntryLevel
boolean lb_AnythingFound = FALSE

```

```

// resolve any derivable levels using the levels (instruction parents)
// specified. Then
// open the search list object so that it will contain any (active)
// instruction for any
// level higher than or equal to the lowest level specified by the caller:
IF of_RetrieveAllApplicable(as_instrumentid, as_customerid,
as_instructiongroup, &
      as_countrytype, as_opbkorgid, ls_parentbkorg) <> gk.SUCCESS THEN
      return gk.FAILURE
END IF

```

```

// See if the list contains an instruction for the current instrument (if
applicable):
IF of_FindForParent(as_InstrucUOID[], as_InstrumentID, as_AddlAttrValue) THEN
  lb_AnythingFound = TRUE
  IF ab_StopWhenFound THEN return gk.SUCCESS
END IF

// If nothing found for instrument or StopWhenFound is FALSE, check if the
list contains
// an instruction for the current customer (if applicable):
IF of_FindForParent(as_InstrucUOID[], as_customerID, as_AddlAttrValue) THEN
  lb_AnythingFound = TRUE
  IF ab_StopWhenFound THEN return gk.SUCCESS
END IF

// If nothing found for customer or StopWhenFound is FALSE, check if the list
contains
// an instruction for the current customer's instruction group:
IF of_FindForParent(as_InstrucUOID[], as_InstructionGroup, as_AddlAttrValue)
THEN
  lb_AnythingFound = TRUE
  IF ab_StopWhenFound THEN return gk.SUCCESS
END IF

// If nothing found for instruction group or StopWhenFound is FALSE, check if
the list contains
// an instruction for the current customer's country:
IF of_FindForParent(as_InstrucUOID[], as_CountryType, as_AddlAttrValue) THEN
  lb_AnythingFound = TRUE
  IF ab_StopWhenFound THEN return gk.SUCCESS
END IF

// If nothing found for country or StopWhenFound is FALSE, check if the list
contains
// an instruction for the current operation bank organization:
IF of_FindForParent(as_InstrucUOID[], as_OpBkOrgID, as_AddlAttrValue) THEN
  lb_AnythingFound = TRUE
  IF ab_StopWhenFound THEN return gk.SUCCESS
END IF

// If nothing found for oper. bank org. or StopWhenFound is FALSE, check if
the list contains
// an instruction for the current operation bank organization's super bank
org:
IF of_FindForParent(as_InstrucUOID[], ls_ParentBkOrg, as_AddlAttrValue) THEN
  lb_AnythingFound = TRUE
END IF

IF lb_AnythingFound THEN
  return gk.SUCCESS
ELSE
  return gk.FAILURE
END IF

```

Object: n_cst_bussrv_assesscharges

Process: 200

Method: of_CreateCharge

Called by n_cst_bussrv_assesscharges from method of_GenerateNewCharges. Creates an Activity Charge component object for the Activity based upon the specified Charge Preload Rule and Charge Instruction.

////////////////////////////////////
 //

/*
 Description:

of_createcharge

Arguments:

any_chrgprldrule	handle to rule that caused this
charge type to preload	
any_chrginstruction	handle to instruction on how to
calculate this charge type	

Returns:

Return SUCCESS or FAILURE

*/
 //////////////////////////////////////
 //

//Revision History

// Name	// Date	Description
// bschuettler	10/01/96	initial release
// bschuettler	04/03/98	skip 'same' charge only if it is not held
(IR#980324170447)		
// BNewman	05/06/98	if the charge amount is zero, then don't create it.
We must		delete the charge. IR9810010925570.

//
 //////////////////////////////////////
 //

// Copyright © 1996 American Management Systems. All Rights Reserved.

//
 //////////////////////////////////////
 //

n_cst_busobj_activitychrg Inv_charge
 n_cst_busobj_fxrate Inv_rate
 n_cst_bussrv_converter Inv_converter

string ls_return, ls_ChrgUoid = '', ls_ChrgCur, ls_InstrumCurrency,
 ls_DistrUoid = '', &
 ls_AN[], ls_AV[], ls_null[], ls_ChrgBasisType, ls_DateRange,
 ls_RateUoid, &
 ls_ChrgBasisAmt, ls_ChargeType, ls_InstrucChargeType, ls_oamt[],
 ls_camt[], &
 ls_RuleUoid, ls_InstrucUoid, ls_DistPtyType[], ls_DistWhen[],
 ls_DistPct[]

decimal ldc_ChrgAmt, ldc_ChrgAmtBase, ldc_ChrgAmtParty, ldc_ChrgBasisAmt[],
 ldc_ChrgBasisAmtConv
 integer li_ChrgBasisDays, i, j, li_numdistributions
 long ll_row[]
 boolean lb_success[]
 decimal ldec_rate
 n_cst_objcls_objectlist Inv_chargelist

// retrieve basic charge data from the preload rule, the instrument and the
 activity:
 any_ChrgPrldRule.of_GetAttribute({'uoid', 'date_range', 'charge_basis_type',
 'charge type'}, ls_av[])
 ls_RuleUoid = ls_av[1]
 ls_DateRange = ls_av[2]

```

ls_ChrgBasisType = ls_av[3]
ls_ChargeType = ls_av[4]
ls_av[] = ls_null[]

// if a charge based on this rule already exists in the activity and it is not
// held, do not preload it again.
// This will be the case if a user has overridden a charge which should not
// produce a 'duplicate'.
inv_chrgelist = inv_activity.of_getcomponenthandle('activity_charge_list')
if inv_chrgelist.of_locate({'related_charge_preload_rule',
'held_charge_activity'}, {ls_ruleuoid, ''}, false, ls_av[], ll_row[]) > 0 then
    return gk.success
end if
ls_av[] = ls_null[]

ls_InstrumCurrency = inv_Instrument.of_GetAttribute('currency')
li_ChrgBasisDays = of_RetrieveChrgBasisDays(inv_Activity, ls_DateRange)
ls_InstrucUoid = anv_ChrgInstruction.of_getattribute('uoid')
of_RetrieveChrgBasisAmt(inv_Activity, ls_ChrgBasisType, ls_ChargeType,
ldc_ChrgBasisAmt[])

// calculate a charge for every charge basis amount found (multiples only when
// a payment
// has multiple doc sets and the 'individual charges' indicator on payment is
// 'no':
for i = 1 to upperbound(ldc_ChrgBasisAmt[])

    ls_ChrgBasisAmt = string(ldc_ChrgBasisAmt[i])

    // figure out who to distribute the charge to based on the instruction and
    // the 'charges for' logic:
    ls_av[] = ls_null[]
    anv_chrginstruction.of_getcomponentattribute('charge_distribution_list',
{'charge_debit_party_type', 'when_collected', 'charge_percentage'}, ls_av[])
    li_numdistributions = upperbound(ls_av[]) / 3
    for j = 0 to li_numdistributions - 1
        ls_DistPtyType[j+1] = ls_av[j*3+1]
        // default any missing party type specifications to the terms' CHARGES
    FOR party type:
        if gnv_app.inv_string.of_iseempty(ls_DistPtyType[j+1]) then
            ls_DistPtyType[j+1] = ls_ChargesForPartyType
            ls_DistWhen[j+1] = ls_av[j*3+2]
            ls_DistPct[j+1] = ls_av[j*3+3]
        end for
        ls_av[] = ls_null[]
        // default missing distributions to one going to the CHARGES FOR party
    type:
        if li_numdistributions = 0 then
            ls_DistPtyType[1] = ls_ChargesForPartyType
            ls_DistWhen[1] = gk.IMMEDIATELY
            ls_DistPct[1] = '100'
            li_numdistributions = 1
        end if

        // Create a charge for every charge distribution object attached to the
        // instruction:
        for j = 1 to li_numdistributions

            // Create a new instance of activity charge in the activity charge list and
            // set its basic data:
            setnull(ls_ChrgUoid)
            ls_Return = inv_Activity.of_NewComponent('activity_charge_list',
ls_ChrgUoid)
            inv_Activity.of_SetComponentAttribute('activity_charge_list(' +
ls_ChrgUoid + ')', &
                {'charge_type', 'charge_basis_amount_s',
'related_charge_preload_rule', 'default_charge_instruction'}, &
                {ls_ChargeType, ls_ChrgBasisAmt, ls_RuleUoid, ls_InstrucUoid},
lb_success[])

            // Set the attributes on the charge's distribution:
            ls_DistrUoid =
inv_Activity.of_GetComponentAttribute('activity_charge_list('+ls_ChrgUoid+')'..c
harge_distribution_list[1]', 'uoid')
            ls_an[] = {'charge_debit_party_type', 'when_collected',
'charge_percentage'}
            ls_av[] = {ls_DistPtyType[j], ls_DistWhen[j], ls_DistPct[j]}

```



```

        inv_Activity.of_SetComponentAttribute('activity_charge_list('+ls_ChrgUoid+
d+').charge_distribution_list('+ls_DistrUoid+'),' ls_an[], ls_av[],
lb_success[])

        // make the charge's formula a copy of the instruction's formula:
        ls_Return = this.of_CopyComponent(anv_ChrgInstruction, inv_Activity, &
            'charge_formula', &

'activity_charge_list('+ls_ChrgUoid+').charge_formula')

        // call calculate on the new charge:
        inv_charge =
inv_activity.of_getcomponenthandle('activity_charge_list('+ls_chrguoid+')')
        inv_charge.of_calculate(this, anv_chrgprldrule)
        // If the charge_amount is zero, then don't create the charge
        if dec(inv_charge.of_getattribute('charge_amount')) = 0 then
            inv_charge.of_delete()
        Else
            ii_NumCharges++
        End If

    end for // distribution loop
end for // charge basis amount loop

return ls_Return

```

Object: n_cst_busobj_activitychrg

Process: 204

Method: of Calculate()

Calls of Calculate on the Charge Formula object and stores the resulting charge.

////////////////////////////////////
 /////

/*
 Description:

of_calculate

Call the calculate responsibility on the charge formula to obtain the latest charge amount. This will be called when the attached formula has changed/is new.

Arguments:

any_chrgservice	handle to the charge
service object	needed to perform needed
functions	
any_rule	handle to the rule based
on which	this charge is to be
calculated	

Returns:

gk.SUCCESS or gk.FAILURE

*/
 //////////////////////////////////////
 /////

//Revision History

// Name	Date	Description
// bschuettler	27 feb 97	initial release
// bschuettler	21 apr 98	IR 971217090227: Make sure internal calculations have 6 digit precision.
// bschuettler	17 aug 98	IR 242: Round charge amounts to their currency's precision.

////////////////////////////////////
 /////

// Copyright © 1996 American Management Systems. All Rights Reserved.

////////////////////////////////////
 /////

string ls_ChrgCur, ls_InstrumentCur, ls_BaseCur, ls_OpBankOrg, ls_RateUoid,
 ls_ChargeType, ls_av[]
 decimal{6} ldc_ChrgBasisAmt, ldc_ChrgBasisAmtConv, ldc_ChrgAmt,
 ldc_ChrgAmtBase, ldc_rate
 integer li_ChrgBasisDays, li_CurrencyPrecision, li_BasePrecision
 s_conversion lstr_conversion

n_cst_busabs_activity lnv_activity
 n_cst_busabs_instrument lnv_instrument
 n_cst_busobj_fxrate lnv_rate
 n_cst_bussrv_converter lnv_converter
 n_cst_codeobj_currency lnv_currency

lnv_Activity = lnv_ImmediateParent
 lnv_instrument = lnv_activity.of_getimmediateparent()
 lnv_instrument.of_getattribute({'base_currency', 'currency'}, ls_av[])
 ls_BaseCur = ls_av[1]
 ls_InstrumentCur = ls_av[2]
 ls_OpBankOrg = lnv_Activity.of_DetermineOperationalBkOrg()
 ls_ChargeType = lnv_AttributeMgr.of_getattribute('charge_type')

of_enableobjectwarehouse(TRUE)
 // get the amount of decimals of all currencies:

```

if inv_objectwarehouse.of_getobject('n_cst_codeobj_currency', ls_BaseCur,
lnv_currency) = gk.success then
    li_BasePrecision =
integer(lnv_currency.of_getattribute('number_of_decimal_places'))
end if
if inv_objectwarehouse.of_getobject('n_cst_codeobj_currency',
ls_InstrumentCur, lnv_currency) = gk.success then
    li_CurrencyPrecision =
integer(lnv_currency.of_getattribute('number_of_decimal_places'))
end if

// get the charge basis amount:
ldc_ChrgBasisAmt =
dec(inv_AttributeMgr.of_GetAttribute('charge_basis_amount_s'))

if isvalid(anv_rule) then
    // Get the amount of days to base this calculation on:
    li_ChrgBasisDays = anv_chrgservice.of_RetrieveChrgBasisDays(lnv_Activity,
anv_rule.of_GetAttribute('date_range'))
else
    // Get the default amount of days to base this calculation on:
    li_ChrgBasisDays = anv_chrgservice.of_RetrieveChrgBasisDays(lnv_Activity,
'')
end if

// Call the formula component to calculate and obtain a new amount:
inv_Formula.of_calculate(ldc_ChrgBasisAmt, ls_InstrumentCur, li_ChrgBasisDays,
ls_OpBankOrg, ldc_ChrgAmt, ls_ChrgCur, ldc_ChrgBasisAmtConv)

// multiply the amount obtained with the percentage of the distribution:
ldc_ChrgAmt = ldc_ChrgAmt *
dec(of_GetComponentAttribute('charge_distribution_list[1]',
'charge_percentage')) / 100

// Convert the amount to base to additionally store the base equivalent:
if of_retrievefxrate(ls_OpBankOrg, ls_ChrgCur, ldc_rate) = gk.success then
    lnv_converter = create n_cst_bussrv_converter
    lnv_converter.of_setrequestor(lnv_activity)
    lstr_conversion.s_operationalorg = ls_OpBankOrg
    lstr_conversion.s_sourcecurrency = ls_ChrgCur
    lstr_conversion.s_targetcurrency = ls_BaseCur
    lstr_conversion.s_origamounts[1] = string(ldc_ChrgAmt)
    lstr_conversion.s_rate = string(ldc_rate)
    if lnv_converter.of_convert(lstr_conversion) <> gk.success then
        destroy lnv_converter
        return gk.failure
    end if
    destroy lnv_converter
    ldc_ChrgAmtBase = dec(lstr_conversion.s_convertedamounts[1])
end if

// set all amount related attributes:
inv_AttributeMgr.of_SetAttribute('calculated_charge_amount_instruction_s',
string(Round(ldc_ChrgAmt, li_CurrencyPrecision)))
inv_AttributeMgr.of_SetAttribute('charge_amount_s', string(Round(ldc_ChrgAmt,
li_CurrencyPrecision)))
inv_AttributeMgr.of_SetAttribute('charge_currency_s', ls_ChrgCur)
inv_AttributeMgr.of_SetAttribute('charge_base_equivalent_amount_s',
string(Round(ldc_ChrgAmtBase, li_BasePrecision)))
inv_AttributeMgr.of_SetAttribute('charge_basis_amount',
string(ldc_ChrgBasisAmt))
inv_AttributeMgr.of_SetAttribute('charge_instruction_basis_amount',
string(ldc_ChrgBasisAmtConv))
inv_AttributeMgr.of_SetAttribute('instruction_currency', ls_ChrgCur)

// copy the waive analysis indicator from the formula:
inv_AttributeMgr.of_SetAttribute('waive_analysis_collect_indicator',
inv_Formula.of_GetAttribute('waive_analysis_collect_indicator'))

// Reset the 'changed' flag on formula:
inv_Formula.of_ResetChange()
return gk.SUCCESS

```

Object: n_cst_busobj_chrgformula
 Process: 206
 Method: of_Calculate()
 Called by of_Calculate on
 n_cst_busobj_activitycharge to calculate the amount
 of the charge.

//

```

/*
  Function:          of_calculate
  Access:           public

  Arguments:
  (input)           adc_ChrgBasisAmt      The amount to base the charge on
  (that of instrument) as_InstrumentCur  The basis amount's currency
  on (input)         ai_ChrgBasisDays     The day range to base the charge
  its currency (output) as_ChrgAmt        The calculated charge amount in
  formula) (output)  adc_ChrgCur         The charge's currency (that of
  formula's currency adc_ChrgBasisAmtConv The charge basis amount in the

  Returns:          string
                   (gk.SUCCESS or gk.FAILURE)

  Description:      Calculates the charge amount for a formula given a
                   CHARGE BASIS AMOUNT / CURRENCY and
                   CHARGE BASIS DAYS.

                   LEGEND:
                   -----
                   C  = Charge Due                      ir_ChargeDue
                   R  = Basic Rate                      ir_BasicRate
                   LRx = Level Rates (x=1-5)            ir_LevelRate[5]
                   LAx = Level Amounts (x=1-5)

ir_LevelAmount[5]  IRx = Interval Rates (x=1-3)
ir_IntervalRate[3] IMx = Interval Minimums (x=1-3)
ir_IntervalMin[3]  BD  = Basis Number of Days for Calculation
ir_ChrgBasisDays   BA  = Basis Amount for Calculation      ir_ChrgBasisAmt
parameter          IGD = Interval Grace Days              system
parameter or formula override
                   DIY = Days in Year                      system
                   LIO = Number of Last Interval Occurences ir_LastIntOccur

                   OVERVIEW:
                   -----
                   INITIALIZE:
                   - Derive the basic rate (R) as follows:
                     IF a multi-level rate specification exists, derive R using
the rate / level table
                     (this could be in tiered or threshold mode)
                     OTHERWISE use the interval basic rate specification. On a
valid formula instance,
                     an interval basic rate will exist only if there is no
rate/level specification.

                   - If an interval specification exists, derive the amount of
times the last/only interval occurs in the
calculation (LIO) based on the overall days (BD) minus any
interval grace days (GD), and if
multiple intervals exist, minus the first interval(s)
which are calculated as occurring once.
                     GD are the number of days into a new interval that should
not be calculated as an interval.

```

```

CALCULATE:
- IF this is an annualized calculation,
  IF it is not a periodic commission, calculate C = R *
  BD / DIY * BA / 100.
  OTHERWISE perform special periodic commission logic
  using associated per. commission details
  END OF CALCULATION
*/
//
// Revision History
//
// Version
// 1.0 Initial version
// RCarden 01 apr 98 Create OpBank Org using object warehouse.
// draum 05 nov 98 IR #980127104925 Rates must have 8 digit
precision
//
//
// Copyright © 1998 American Management Systems, Inc. All Rights Reserved.
//
//
n_cst_busabs_Activity Inv_Activity
n_cst_bussrv_AssessChrgs Inv_AssessChargesSrv
n_cst_busobj_chrgprldrulelc Inv_Rule
n_cst_busobj_OpBankOrganization Inv_opbkorg

boolean lb_CalcSubstituteBasisAmount
string ls_ChargeType, ls_CurrencyInd, ls_SubstituteChargeBasisType
s_conversion lstr_conversion

idc_ChrgBasisAmt = adc_ChrgBasisAmt
ii_ChrgBasisDays = ai_ChrgBasisDays

Inv_Activity = inv_ImmediateParent.of_GetImmediateParent()

// Initialize Level/Interval calculation specifications and the BASIC RATE:
of_Initialize()

// determine the resulting charge's currency using the indicator on operation
bank organization:
// This has been disabled due to a pending issue (burkhard 4/15/98)
// if not ib_flat then
if false then
  of_EnableObjectWarehouse(True)
  if inv_ObjectWarehouse.of_GetObject('n_cst_busobj_opbankorganization',
as_opbankorg, Inv_opbkorg) = gk.success then
    ls_currencyind =
Inv_opbkorg.of_getattribute('charge_currency_indicator')
  else
    ls_currencyind = gk.currency_of_instruction
  end if

  choose case ls_currencyind
    case gk.base_currency
      as_ChrgCur = is_basecurrency
    case gk.currency_of_instrument
      as_ChrgCur = as_instrumentcur
    case gk.currency_of_instruction
      as_ChrgCur = Inv_AttributeMgr.of_GetAttribute('currency')
  end choose
else
  as_ChrgCur = inv_AttributeMgr.of_GetAttribute('currency')
end if

// (For interval charges only) The full instrument liability or outstanding
// amount should be used to calculate the charge due rather than the delta
// amount -- for intervals created during an amendment because of an extension
// in the end date. Calculate the substitute basis amount for use in these
cases.
if is_CalcType = 'INT' then
  // get the charge basis type from the related rule of the charge
  Inv_Rule = CREATE n_cst_busobj_chrgprldrulelc
  Inv_Rule.of_Open(inv_ImmediateParent.DYNAMIC
of_GetAttribute('related_charge_preload_rule'), &
is_UserID, is_Password, gk.ACCESS_READ)

```

```

ls_SubstituteChargeBasisType =
Inv_Rule.of_GetAttribute('charge_basis_type')
DESTROY Inv_Rule

// ...and be sure to use the full amount instead of the delta...
if (ls_SubstituteChargeBasisType = gk.BANK LIABILITY_AMOUNT_CHANGED) then
  ls_SubstituteChargeBasisType = gk.BANK LIABILITY_AMOUNT
  lb_CalcSubstituteBasisAmount = TRUE
elseif ls_SubstituteChargeBasisType = gk.OUTSTANDING_AMOUNT_CHANGED then
  ls_SubstituteChargeBasisType = gk.OUTSTANDING_AMOUNT
  lb_CalcSubstituteBasisAmount = TRUE
else
  lb_CalcSubstituteBasisAmount = FALSE
end if

// Now, if we need to recalculate the charge basis amount, create the
service object and retrieve the amount...
if lb_CalcSubstituteBasisAmount then
  decimal{6} ldc_temp[] // since the function takes an array but we only
want one...

  ls_ChargeType = inv_ImmediateParent.DYNAMIC
of_GetAttribute('charge_type')

  Inv_AssessChargesSrv = CREATE n_cst_bussrv_AssessChrgs
  Inv_AssessChargesSrv.of_RetrieveChrgBasisAmt(Inv_Activity,
ls_SubstituteChargeBasisType, ls_ChargeType, ldc_Temp[])
  ldc_SubstituteBasisAmount = ldc_Temp[1]
  DESTROY Inv_AssessChargesSrv
end if
else
  // For non-interval charges, we don't want a substitute basis amount...
  lb_CalcSubstituteBasisAmount = FALSE
end if

// if the currency of the charge/formula is different than that of the
instrument, convert
// the charge basis amount to the charge currency and substitute basis amount
to the formula currency.
if as_ChrgCur <> as_InstrumentCur then
  n_cst_busobj_fxrate Inv_rate
  n_cst_bussrv_converter Inv_converter
  string ls_RateVoid
  decimal{8} ldc_rate
  // if the charge currency is base, get the instrument currency rate, else
the charge currency rate:
  if as_ChrgCur = is_BaseCurrency then
    inv_immediateparent.dynamic of_retrievefxrate(as_OpBankOrg,
as_InstrumentCur, ldc_rate)
  else
    inv_immediateparent.dynamic of_retrievefxrate(as_OpBankOrg, as_ChrgCur,
ldc_rate)
  end if
  // now convert from instrument currency to formula currency using the found
rate:
  Inv_converter = create n_cst_bussrv_converter
  Inv_converter.of_setrequestor(this)
  if ldc_rate > 0 then
    // set up the converter...
    lstr_conversion.s_operationalorg = as_OpBankOrg
    lstr_conversion.s_sourcecurrency = as_InstrumentCur
    lstr_conversion.s_targetcurrency = as_ChrgCur
    lstr_conversion.s_rate = string(ldc_rate)
    // ...then convert the charge basis amount...
    lstr_conversion.s_origamounts[1] = string(idc_ChrgBasisAmt)
    if Inv_converter.of_convert(lstr_conversion) <> gk.success then
      ldc_ChrgBasisAmt = 0
    else
      ldc_ChrgBasisAmt = dec(lstr_conversion.s_convertedamounts[1])
    end if
  // ...and if necessary, then the substitute basis amount...
  if lb_CalcSubstituteBasisAmount then
    lstr_conversion.s_origamounts[1] = string(idc_SubstituteBasisAmount)
    if Inv_converter.of_convert(lstr_conversion) <> gk.success then
      ldc_SubstituteBasisAmount = 0
    else

```

```

        idc_SubstituteBasisAmount =
dec(lstr_conversion.s_convertedamounts[1])
    end if
    end if
else
    // ...if the conversion rate is 0, then the amount is 0 also...
    idc_ChrgBasisAmt = 0
    // ...as is the substitute amount...
    if lb_CalcSubstituteBasisAmount then idc_SubstituteBasisAmount = 0
    end if
    destroy lnv_converter
end if

// pass the basis amount used back to the caller for their reference:
adc_ChrgBasisAmtConv = idc_ChrgBasisAmt

IF ib_Annualized THEN idc_BasicRate = idc_BasicRate * ii_ChrgBasisDays /
ii_DaysInYear

CHOOSE CASE is_CalcType
CASE 'FLT'
    idc_ChargeDue = dec(lnv_AttributeMgr.of_GetAttribute('flat_amount'))
CASE 'PER'
    idc_ChargeDue = 0
    of_adjusttominmax(idc_ChargeDue)
CASE 'RTE', 'LVL'
    idc_ChargeDue = idc_BasicRate * idc_ChrgBasisAmt / 100
    of_adjusttominmax(idc_ChargeDue)
CASE 'INT' //Interval charge
    // perform interval-specific initialization...
    if of_InitializeIntervalVars() <> gk.SUCCESS then
        return gk.FAILURE
    end if
    // ...then create (or update) the charge intervals...
    if of_CreateOrUpdateIntervals() <> gk.SUCCESS then
        return gk.FAILURE
    end if
    // ...and finally, calculate the amounts for each interval and the total
amount
    idc_ChargeDue = of_CalcIntervalCharge()
END CHOOSE

adc_ChrgAmt = idc_ChargeDue
return gk.SUCCESS

```